COMPUTER SCIENCE
TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

(9) Technical rept., (1)

(14) TR-541
DAAG53-76C-0138

(11) June 77

(12) 21 p.

(6) CONSTRUCTING TREES FOR REGION DESCRIPTION.

(10) David L. Milgram
Computer Science Center
University of Maryland
College Park, MD 20742

ABSTRACT

A thresholded (binary) image can be represented as a region tree in which each node corresponds to a component of 1's (object) or 0's (background). If regions O and B share a border, then one encloses the other. This enclosure relation defines the tree. The chain code of the component and a description based on statistical features are stored at each node. The region tree is built in a single pass over the image.

(15) The support of the U. S. Army Night Vision Laboratory under Contract DAAG53-76C-0138, ARPA Order-3206 is gratefully acknowledged, as is the help of Mrs. Shelly Rowe.

(6)

403018

1473

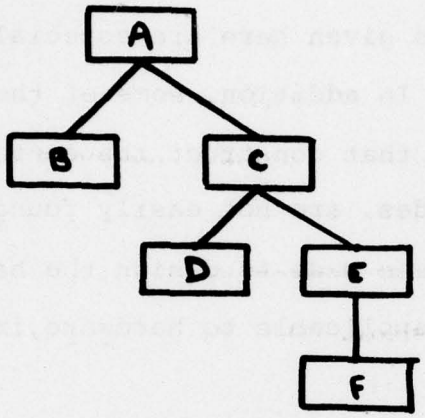
1. Introduction

The algorithms to be described here are basically well known. However, the versions given here are especially simple and well-structured. In addition, some of the algorithms, particularly those that construct the containment tree and the border chain codes, are not easily found in the literature. An effort has been made to design the basic algorithmic steps in a manner applicable to hardware implementation.

We assume that we are given an image and a threshold. The border of the image consists of below-threshold points. For each connected component, we wish to extract and store three essentially different kinds of information. The first consists of descriptive features over the region such as mean and standard deviation of gray level, area, perimeter, moments, etc. The second type of information is a chain encoding of the outer boundary of the region. Note that while a region may have many holes (each corresponding to an inner boundary) it has exactly one outer boundary. The third type of information is stored inherently by the data structure and corresponds to the containment relations among regions. It is convenient to store the complete set of connected components, including components of background, rather than simply the above-threshold regions.

As an example of the desired structure, consider the following region map:

The data structure describing the regions of this map is:



Each node contains the feature data and the boundary encoding of the named region. Note that if the root is considered to be at level 0 then nodes at odd levels correspond to above-threshold regions while nodes at even levels (>0) correspond to holes.

In addition to producing a data structure the algorithm can produce a labelled image in which each connected component is assigned a unique label. The ability to know precisely which pixels belong to the same component is very useful for display as well as analysis. Labelled regions thought to be of particular interest can be highlighted or outlined; while regions of no interest (e.g., clutter, noise, accidents) can be suppressed in the display. However, while the data structure can be produced in a single pass over the original image, it requires two passes to produce the labelled region map: the first, to partially label the image and create a label equivalence table; the

second, to read both the partially labelled region map and the label equivalence table and to produce the final region map.

The connectivity decisions which make up the core of the algorithm use a 2x2 processing window. However, a 3x3 window is used since it also allows statistics based on border points to be computed. The processing window moves across the image in raster fashion so that every point of the image is in the center of the 3x3 window exactly once. It is convenient to embed the image in a row of background levels which are not processed directly. This allows us to define the algorithm without considering boundary conditions. We represent the pixels in a 3x3 window as

```
abc
dxe
fgh
```

. Thus three rows of the image are maintained at all times (although, as we mentioned, only two rows are strictly necessary). In addition, various auxiliary data structures are needed.

Two algorithms will be presented. The first describes region tracking; the second constructs boundary chain codes. It is straightforward to integrate the two algorithms into a single pass.

2. Region tracking

This algorithm constructs for each connected component a descriptive feature vector. Although we do not specify the features, it is assumed that they are all extractable from a raster scan using a 3x3 processing window. Additional storage is available to hold the feature values for the components.

When a new region is encountered during a raster scan, it is assigned a vector of registers to store its feature values. As the region is being tracked on the same row or continued on the next row, values continue to be accumulated into its feature vector. In order to specify the correspondence between a region and its register vector, a label is created and assigned to each point of the region which has already been visited. The label will identify the appropriate register vector, usually by some indexing scheme. Region points found to be adjacent to already labelled region points inherit that label value and contribute statistically to its register vector.

Often a region encountered for what is thought to be the first time may at a later row prove to be connected to a previously encountered region. Such regions are called subcomponents. Inasmuch as feature values were being maintained separately for each subcomponent, it becomes necessary to combine the feature values immediately or at least to create a flag that signifies that the values for the two subcomponents will eventually be combined. If the

latter course is chosen, a table is constructed indicating which labelled regions overlap. The label equivalence table can be stored either as a bit matrix or as a list. In this algorithm we take the immediate approach and combine feature vectors for equivalent regions as soon as the equivalence is discovered. The fact of recombination imposes constraints on the types of features which can be evaluated during the single pass. Thus while the area of a component is the sum of areas of its subcomponents, it is not possible to compute the maximum row extent of a component from the run widths of its subcomponents. This feature, of course, could be computed on a second pass (i.e., over the labelled region map).

Since region labels propagate from point to point, we must maintain the labels of the current line and the previous line as well as their gray levels. Actually, only the labels of those points in the preceding row that are neighbors of unexamined points in the current row, and the labels of those examined points in the current row, need be stored. Thus the amount of storage necessary for labelled points can be reduced to a single row.

Just before a new row is read in to be processed, the current row becomes the previous row. Rather than copy the data from one data area to another, one ordinarily cycles an indirect pointer array. However, in hardware, copying the data may be more efficient than cycling a pointer since the copying can be done in parallel.

The label assigned to a component should designate whether the component is above or below threshold. If the background is not partitioned into regions (i.e., is ignored) by the algorithm then the data structure becomes simply a list of above-threshold regions. This is suitable for many applications, e.g., infrared target cueing. For the sake of generality, we describe below how to determine the containment relation which defines the tree structure. It is evident that if two components of a binary image are adjacent then one encloses the other. However, if more than one object-background transition has been detected, one cannot know which encloses which from strictly local information at the time of initial label assignment. The determining condition is "which region terminates first"? The region terminating first is enclosed by the adjacent region. Thus whenever a region terminates, the data structure is updated to reflect the containment relation. When a region is initiated it is entered onto an "active" list -- the list of unterminated regions. At the end of each row, the active list is compared with the list of component labels of the current label row. Any active component whose label does not appear in the current label row must have terminated. Additionally, when overlapping regions are combined, the discarded label must be deleted from the active list.

For simplicity, the algorithm will use 0 to indicate a point below threshold and 1 to indicate a point above

For example, $\begin{matrix} ab \\ dx \end{matrix} = \begin{matrix} 00 \\ 11 \end{matrix}$ indicates that only d and x are above threshold. We are also assuming that regions of above threshold points are 4-connected (i.e., based on adjacency of 4-neighbors). To maintain consistency when tracking the background, we require that background regions be defined by 8-connectivity.

Other implementations of this algorithm operate by maintaining lists of endpoints of above threshold runs. This can increase the efficiency of the algorithm when computing descriptive features, since for features not dependent on the gray level, one can compute the total contribution to the feature accumulator based on the run rather than at each individual point of the run. This can result in significant savings. The above algorithm can be modified to take advantage of this saving.

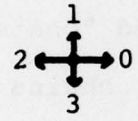
Algorithm for Producing the Labelled Region Tree

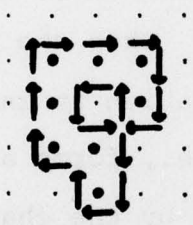
```
"INITIALIZE"
   REGION_COUNT ← 1
   REGION_NOT_NEW ← FALSE
   NEW_LABEL_NEEDED ← FALSE
   PLACE BACKGROUND LABEL ON ACTIVE LIST
   DO  FOR X = 1,...,WIDTH "Label the outer boundary as background"
       CURRENT_LABELS(X) ← <REGION_COUNT,BACKGROUND>          OD

"PROCESS EACH NEW IMAGE RECORD"
   DO  FOR EACH IMAGE RECORD
       DO  FOR X = 1,...,WIDTH "Cycle the label buffers"
           PREVIOUS_LABELS(X) ← CURRENT_LABELS(X)             OD
       READ AN IMAGE RECORD AND CYCLE THE IMAGE ROW BUFFERS
       DO  FOR X = 1,...,WIDTH "Threshold and label each pixel"
           CALL DETERMINE_LABEL(X,LABEL,NEW_LABEL_NEEDED,REGION_NOT_NEW)
           CURRENT LABELS(X) ← LABEL
           IF  NEW_LABEL_NEEDED
               THEN  "We seem to be entering a new region"
                     PLACE LABEL ON ACTIVE LIST
                     NEW_LABEL_NEEDED ← FALSE        FI
           IF  REGION_NOT_NEW
               THEN  "Merge the two equivalent region descriptions"
                     CALL EQUIV(PREVIOUS_LABELS(X),CURRENT_LABELS(X-1))
                     REGION_NOT_NEW ← FALSE          FI
           CALL DOSTATS(X) "Accumulate the feature data for X"
       OD
       DO  FOR EACH LABEL ON THE ACTIVE LIST BUT NOT IN CURRENT_LABELS
           "Delete any completed region from the active list"
           FIND LEAST X SO THAT PREVIOUS_LABELS(X) = LABEL
           PARENT_LABEL ← CURRENT_LABELS(X)
           DELETE LABEL FROM ACTIVE LIST.  PLACE ON COMPLETED LIST.
           APPEND LABEL TO THE CONTAINMENT LIST OF PARENT_LABEL
       OD
   OD
```

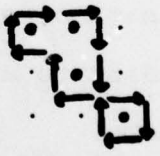
```
"At this point the completed region list has the required tree
 structure"
"Now compute the desired features from the accumulated feature
 data at each node"
RETURN
END


PROCEDURE DETERMINE_LABEL(POSN,LABEL,NEW,EQ)
"Assign a label to the lower right position of a 2x2 neighborhood"
COMPUTE C = VECTOR OF THRESHOLD DECISIONSFOR A 2x2 NEIGHBORHOOD AT POSN.
CASE C OF

   |00  01  11|
   |00, 10, 11|

        DO  "Label the point with the upper-left label"
            LABEL ← PREVIOUS_LABELS(POSN-1)

   |00  10  11  01|
   |10, 10, 01, 01|

        DO  "Label the point with the upper-right label"
            LABEL ← PREVIOUS_LABELS(POSN)

   |01  11  10  00|
   |00, 00, 11, 11|

        DO  "Label the point with the lower-left label"
            LABEL ← CURRENT_LABELS(POSN-1)

   |10  01|
   |00, 11|

        DO  "Equivalence the lower-left and upper-right labels"
            LABEL ← PREVIOUS_LABELS(POSN)
            EQ ← TRUE

   |11|
   |10|

        DO  "Create a new background label"
            REGION_COUNT ← REGION_COUNT+1
            LABEL ← <REGION_COUNT,BACKGROUND>
            NEW ← TRUE
```

```
|00|
|01|

  DO  "Create a new foreground label"
      REGION_COUNT ← REGION_COUNT+1
      LABEL ← <REGION_COUNT,FOREGROUND>
      NEW ← TRUE

|10|
|01|

  DO  "There is background equivalence and a new foreground label"
      REGION_COUNT ← REGION_COUNT+1
      LABEL ← <REGION_COUNT,FOREGROUND>
      NEW ← TRUE
      EQ ← TRUE

ESAC
RETURN
END


PROCEDURE EQUIV(LABEL1, LABEL2)

"Two region labels identify the same region"
IF  LABEL1 ≠ LABEL2
    THEN  "Merge the two region descriptions"
          COMBINE THE ACCUMULATED STATISTICS OF LABEL2 WITH LABEL1
          APPEND THE CONTAINMENT LIST OF LABEL2 to LABEL1
          DELETE LABEL2 FROM THE ACTIVE LIST
          DO  FOR X = 1,...,WIDTH
              IF  PREVIOUS_LABELS(X) = LABEL2
                  THEN  PREVIOUS_LABELS(X) ← LABEL1      FI
              IF  CURRENT_LABELS(X) ≈ LABEL2
                  THEN  CURRENT_LABELS(X) ← LABEL1       FI
          DO

FI
RETURN
END
```

3. Constructing boundary chains

This subsection describes that portion of the overall algorithm for region description which creates and stores coded representations of boundaries. As was said before, each region has exactly one outer boundary. The encoding of the boundary uses a more primitive code than the Freeman code. However, it is trivial to convert the chain code given here. The boundary chain of a region is defined here to follow the cracks between two regions using four directions: 

. We illustrate the boundary encoding for a simple region:  Note that this region does not have a hole and that if one tracks the boundary by keeping the right hand on the object and taking right turns when possible then the encoding consists of a single trace. On the other hand, consider:



Following the previous tracing rule, one discovers two disjoint boundary encodings. The encoding we have described defines 4-connected foreground regions. 8-connected region boundaries are defined by switching the roles of 0's and 1's.

The reason for choosing a crack-following scheme rather

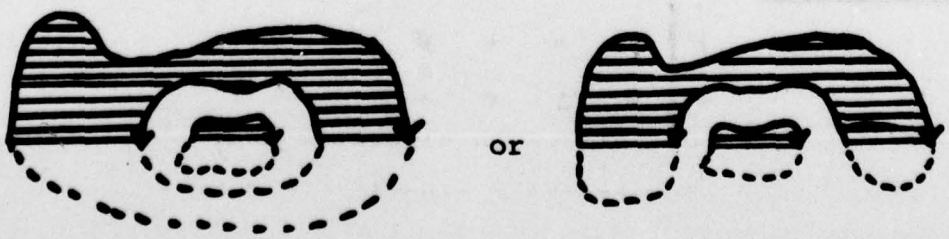
than a typical 8 direction code is to ensure that no two region boundaries intersect. The encodings themselves are somewhat less economical and presumably the algorithm could be modified suitably to accommodate a more efficient encoding.

As before, the input image is processed in a raster scan based on the same 2x2 neighborhood window (within the 3x3 processing window). Since many objects may intersect the row currently being processed, a list of chain segments, called the active chain list, is maintained. Chain links arising out of a 2x2 neighborhood are called "chainlets". Chainlets known to extend existing (active) chains are concatenated to those chains. Thus the active chain list is a list of strings. Concatenation is indicated by an ampersand. When a chain is closed, i.e., forms a cycle, then we know that the object surrounded by the chain is terminated. Alternatively, if a chain has not been extended in the current row then it must have terminated in the previous row. This test for termination can be synchronized with the termination test for regions so that the tree node corresponding to a region and its boundary can be closed at that time.

When a chainlet is discovered to be both the head of one active chain and the tail of another, we concatenate the two active chains into a single chain -- or, if the two chains are the same already, terminate the chain. The overall structure of this chain algorithm thus is quite similar to the region tracking algorithm and both can be incorporated within the same overall framework.

Consider the 2x2 neighborhood $\begin{matrix} a & b \\ d & x \end{matrix}$ for the point x. Let the image coordinates of x be (i,j). The chainlet $\begin{matrix} a & b \\ d\uparrow & x \end{matrix}$ is denoted (i+1, j→i, j); similarly, $\begin{matrix} a & b \\ d & \rightarrow \\ & x \end{matrix}$ gives rise to (i, j→i, j+1). An open chain can be represented by a head and a tail, each specified by a pair of image coordinates and a sequence of moves from head to tail based on the previous 4-direction code. In a closed chain, the head and tail and the same coordinate pair. The point chosen to "anchor" a closed chain can be any point through which the chain passes. A good choice for anchor point might be the rightmost point on the bottommost row, since this is the last point seen by the algorithm.

When a chain is terminated, we associate the chain encoding with the tree node corresponding to the labelled region enclosed by the chain. One cannot determine at the time of termination which active chain will ultimately contain the terminated chain. It is certainly one of the active chains (if we include the outer boundary of the image as a vacuous chain), but as the figure below indicates there is not enough information to determine which. It is



It is therefore desirable that boundary encoding be embedded within the region tracking algorithm so that a complete tree description can be produced.

The algorithm as presented is linear in the number of points. However, for the configurations $\{\begin{smallmatrix}00\\00\end{smallmatrix}, \begin{smallmatrix}11\\11\end{smallmatrix}\}$ nothing need be done. Thus the algorithm is actually linear in the number of boundary points, which may be a significantly smaller set of points. The operations FIND_HEAD_AT and FIND_TAIL_AT can be coded as binary searches on the column index, or even more efficiently within a hash table.

The four-direction encoding is easily converted to a Freeman chain encoding. We describe the translation for the boundary chain of a object. Boundaries of holes are treated similarly. Each segment (codon) of the four-direction encoding is replaced by <u>at most</u> one Freeman codon by considering that codon and its forward neighbor. The table below shows the translation matrix. The symbol ∅ indicates the null code; while Δ indicates that this combination cannot occur in the four-direction encoding.

current codon \ next codon	↑	→	↓	←
↑	↑	∅	Δ	↖
→	↗	→	∅	Δ
↓	Δ	↘	↓	∅
←	∅	Δ	↙	←

Algorithm to Construct Boundary Chains

```
"INITIALIZE"
   SET THE ACTIVE CHAIN LIST TO EMPTY
"PROCESS EACH IMAGE RECORD"
   DO  FOR EACH IMAGE RECORD, I = 1,...,LEGNTH
       READ AN IMAGE RECORD AND CYCLE THE INPUT BUFFERS
       DO  FOR J = 1,...,WIDTH
           CALL PROCESS_CHAINLET (I,J)
       OD
   OD
   STOP
   END


PROCEDURE PROCESS_CHAINLET(I,J)
  LET C BE THE 2x2 BINARY RESULT OF THRESHOLD   a b
                                                d x
  LET COLOR BE THE RESULT OF THRESHOLDING X
  LET COLORA BE THE RESULT OF THRESHOLDING A
  CASE C OF

      |00   11|
      |00,  11|
            DO  "Nothing"

      |00,  11|
      |01,  10|
            DO  CALL CREATE(I,J,COLOR)

      |10|
      |01|
            DO  CALL CREATE(I,J,COLOR)
                CALL MERGE(I,J,COLORA)

      |10   01|
      |00,  11|
            DO  CALL MERGE(I,J,COLORA)
```

```
      |00  10  00  10|
      |10, 10, 11, 11|

                    DO  CALL AFFIX(I,J,COLOR)

      |01|
      |10|

                    DO  CALL AFFIX(I,J,COLOR)
                        CALL APPEND(I,J,COLOR)

      |01  11  01  11|
      |00, 00, 01, 01|

                    DO  CALL APPEND(I,J,COLOR)

ESAC
RETURN
END

PROCEDURE MERGE(I,J,COLOR)

  CALL FIND_TAIL_AT(I,J,CHAIN1)
  CALL FIND_HEAD_AT(I,J,CHAIN2)
  IF  K = L
      THEN  CALL COMPLETE_CHAIN(CHAIN1,COLOR)
            DELETE CHAIN1 FROM ACTIVE LIST
      ELSE  TAIL(CHAIN1) ← TAIL(CHAIN2)
            CHAIN(CHAIN1) ← CHAIN(CHAIN1) & CHAIN(CHAIN2)
            DELETE CHAIN2 FROM ACTIVE CHAIN LIST
  FI
  RETURN


PROCEDURE AFFIX(I,J,COLOR)

  CALL FIND_HEAD_AT(I,J,CHAIN1)
  IF  COLOR = BACKGROUND
      THEN  HEAD(CHAIN1) ← (I+1,J)
            CHAIN(CHAIN1) ←  "↓" & CHAIN(CHAIN1)
      ELSE  HEAD(CHAIN1) ← (I,J+1)
            CHAIN(CHAIN1) ←  "→" & CHAIN(CHAIN1)
  FI
  RETURN
```

```
PROCEDURE APPEND(I,J,COLOR)
  CALL FIND_TAIL_AT(I,J,CHAIN1)
  IF  COLOR = BACKGROUND
      THEN  TAIL(CHAIN) ← (I,J+1)
            CHAIN(CHAIN1) ← CHAIN(CHAIN1) & "←"
      ELSE  TAIL(CHAIN1) ← (I+1,J)
            CHAIN(CHAIN1) ← CHAIN(CHAIN1) & "↑"
  FI
  RETURN


PROCEDURE FIND_TAIL_AT(I,J,CH)
   "Note:  This routine is guaranteed to succeed"
   FIND CH SO THAT TAIL(CH) = (I,J)
   RETURN


PROCEDURE FIND_HEAD_AT(I,J,CH)
   "Note:  This routine is guaranteed to succeed"
   FIND CH SO THAT HEAD(CH) = (I,J)
   RETURN


PROCEDURE COMPLETE_CHAIN(CH,COLOR)
   "Note:  Head (CH) is the lower right corner of the completed
      chain.  Color is the color of the top right point of the
      2x2 neighborhood"
    IF  COLOR = BACKGROUND
        THEN  "CH is the boundary of a hole"
        ELSE  "CH bounds a foreground region"
   FI
   ASSIGN CHAIN DESCRIPTION TO REGION DESCRIPTION FOR THE REGION
   RETURN
   END
```

4. Conclusion

The algorithms presented here allow the user to construct a tree description of the regions contained in a thresholded image. The arc relation defining the tree is one of containment while the information stored at each node consists of a statistical description of the region and a chain encoding of its boundary.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* CONSTRUCTING TREES FOR REGION DESCRIPTION		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER TR-541
7. AUTHOR(s) David L. Milgram		8. CONTRACT OR GRANT NUMBER(s) DAAG53-76C-0138
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Ctr. Univ. of Maryland College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Night Vision Lab. Ft. Belvoir, VA 22060		12. REPORT DATE June 1977
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Connected component tracking
Chain coding
Tree structure
Image processing

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A thresholded (binary) image can be represented as a region tree in which each node corresponds to a component of 1's (object) or 0's (background). If regions O and B share a border, then one encloses the other. This enclosure relation defines the tree. The chain code of the component and a description based on statistical features are stored at each node. The region tree is built in a single pass over the image.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*